ASSIST2CCM User Manual

Sonia Campa, Massimo Coppola,
Silvia Magini, Laura Potiti, Diego Puppin

September 2004

Table of Contents

1 Installing OpenCCM 0.8.1 and OpenORB 1.4.0.BETA1....3
1.1 Installing OpenCCM 4
1.2 Installing OpenORB....5
1.3 Cluster pianosa and Cluster c1....5
2 Installing and Using ASSIST2CCM 1.0....6
2.1 The TASKCCM directory....7
2.1.1 Available Scripts....8
2.1.1.1 The env_CCM Script....8
2.1.1.2 Scripts to Compile Component IDL3 and Java Files....9
2.1.1.3 Scripts to Launch and Stop the CCM Application....9
2.1.1.4 The assist2ccm.sh Script....10
2.1.2 Tested Programs (assist/solo_assist/seq)....10
2.1.3 The CDR_IO and CDRServer classes (socket)....10
2.1.4 The XML Parser (parserXML)....11
2.1.5 The demo Directory and Archived Tests (demo, demo/archivio)....11
3 Using ASSIST2CCM....11
4 Some Example CCM Applications: Regression Tests....13
4.1 REGRESSION_1IN....13
4.2 REGRESSION_1IN_1OUT....13
4.3 REGRESSION_2IN_1OUT....14
4.4 REGRESSION_2IN_2OUT....14
5 Performance 15
5.1 Performance: Events....15
5.2 Performance: RMI....17

List of Figures

Figura 1. Structure of the ASSIST2CCM_1.0 tree. 6
Figure 2. Directories created within the demo sub-tree by assist2ccm.sh. 12
Figure 3. Structure of the test CCM application. 15
Table 4. Sequential timing of the benchmark. 15
Figure 5. Timing of the CCM application. 16
Figure 6. Speed-up of the CCM application. 16
Figure 7. Bandwidth with the event implementation. 16
Figure 8. Structure of the CCM application. 17
Table 9. Sequential timing of the benchmark. 17
Table 10. Timing of the test CCM application. 18
Table 11. Speed-up of the CCM application. 18

1 Installing OpenCCM 0.8.1 and OpenORB 1.4.0.BETA1

In the following, we assume *bash* shell is used, and that JAVA is installed in the directory */usr/java*. The version we tested is JAVA 1.4.2_03, which is available at http://java.sun.com/.
First, set the JAVA_HOME environment variable to the right path, then add the Java compiler path to the PATH variable, as follows:

```
export JAVA_HOME=/usr/java/j2sdk1.4.2_03
export PATH=$PATH::$JAVA_HOME/bin
```

Download the OpenCCM installation package from the OpenCCM site http://openccm.objectweb.org/. You need *OpenCCM-0.8.1.tar.gz*.

From the OpenORB site (http://openorb.sourceforge.net/), another three packages are required:

1. *OpenORB-1.4.0-BETA1.tgz*
2. *NamingService-1.4.0-BETA1.tgz*
3. *Tools-1.4.0-BETA1.tgz*

Unpack them in a common directory, such as */l/disc1*, and then set the TCOO_HOME variable to the path where OpenORB and its tools can be found.

```
cd /l/disc1

tar -xzvf OpenCCM-0.8.1.tar.gz
tar -xzvf OpenORB-1.4.0-BETA1.tgz
tar -xzvf NamingService-1.4.0-BETA1.tgz
tar -xzvf Tools-1.4.0-BETA1.tgz

export TCOO_HOME=/l/disc1
```

Four directories will be created:

- OpenCCM-0.8.1
- OpenORB
- NamingService
- tools

1.1 Installing OpenCCM

Set the CCM_HOME variable to point to the path where OpenCCM was installed:

```
export CCM_HOME=/l/disc1/OpenCCM-0.8.1
```

Then, enter the *$CCM_HOME* directory and

1. Run *build.sh* to create a file named *build.properties*.

```
build.sh
```

2. Modify *build.properties*, just created: uncomment the row with the name of the used ORB (*ORB.name*), and specify the absolute path of the ORB (*ORB.home.dir*):

```
Changes to build.properties:

1) uncomment the line: ORB.name=OpenORB-1.4.0

2) specify: ORB.home.dir=/l/disc1/OpenORB/
```

3. Run *build.sh install*:

```
build.sh install
```

4. Run again *build.sh* to compile OpenCCM:

```
build.sh
```

5. Run again *build.sh install* to perform the final installation of OpenCCM:

```
build.sh install
```

Last, three more variables need to be set properly.

1. The *OpenCCM_HOMEDIR* variable is set to the directory where CCM executables can be found:

```
export OpenCCM_HOMEDIR= $CCM_HOME/openccm/build
```

2. The *OpenCCM_CONFIG_DIR* variables tells where temporary files (created during the process of compiling and running CCM programs) must be stored. This temporary directory is created and removed at each compilation or run.

```
export OpenCCM_CONFIG_DIR=$CCM_HOME/openccm/tmp
```

3. The path of CCM executable files must be added to the *PATH*:

```
export PATH=$PATH:$CCM_HOME/openccm/build/bin
```

1.2 Installing OpenORB

OpenORB is distributed in precompiled packages (JAR Java Archives), and is ready to be used. The only manual intervention is moving two JARs from the NamingService installation directory to the OpenORB tree.

```
cp NamingService/lib/openorb_tns-1.4.0.jar                OpenORB/lib
cp NamingService/lib/endorsed/openorb_ns_omg-1.4.0.jar    OpenORB/lib/endorsed/
```

Then, we need to set two environment variables:

1. The *OpenORB_LIB* variable tells where OpenORB JARs are stored:

```
export OPENORB_LIB=$TCOO_HOME/OpenORB/lib
```

2. *TOOLS* is used as a pointer to OpenORB tool JARs:

```
export TOOLS=$TCOO_HOME/tools/lib
```

1.3 Cluster *pianosa* and Cluster *c1*

In the following, when describing the installation scripts and commands, we will refer to two different clusters: *pianosa* and *c1*. Their configurations are slightly different:

- on cluster *c1*, we have OpenCCM 0.8.0 and OpenORB 1.3.1.
- on cluster *pianosa*, we have OpenCCM 0.8.1 and OpenORB 1.4.0.BETA1.

2 Installing and Using ASSIST2CCM 1.0

Figure 1 describes the overall architecture of the ASSIST2CCM_1.0 tool. The software suite is organized into sub-directories as follows.

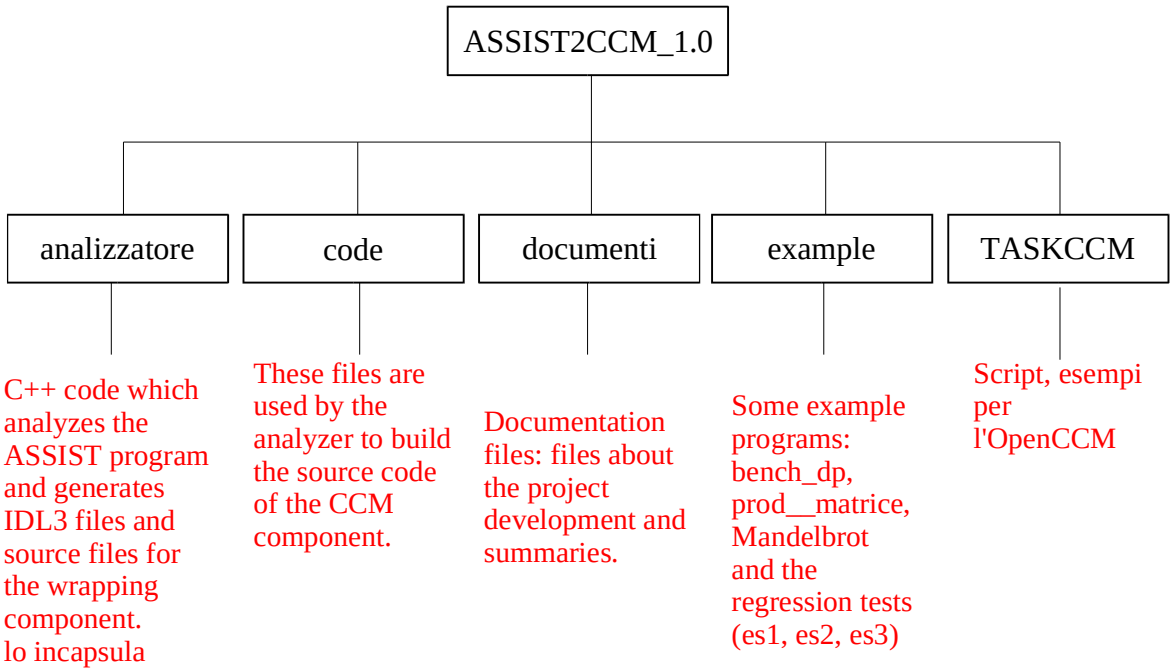
ASSIST2CCM_1.0
analizzatore
code
documenti
example
TASKCCM
C++ code which analyzes the ASSIST program and generates IDL3 files and source files for the wrapping component. lo incapsula
These files are used by the analyzer to build the source code of the CCM component.
Documentation files: files about the project development and summaries.
Some example programs: bench_dp, prod__matrice, Mandelbrot and the regression tests (es1, es2, es3)
Script, esempi per l'OpenCCM

Figura 1. Structure of the ASSIST2CCM_1.0 tree.

The directories are so organized:

1. *Analizzatore (analyzer)*
   Here the user will find the source code of our ASSIST analyzer, which is described in detail in the document "ASSIST2CCM:Implementation". To install this tool, set the *ASSIST2CCM* environment variable to point to this directory and perform *make*.

2. *Code*
   This directory stores some files used by the analyzer to build the souce code of the wrapping CCM component.

3. *Documenti (documents)*
   Here, the user will find the documents called TASK_CCM_DOC (1-11), describing the step-by-step evolution of this project, and some summary documents. These documents are written in Italian, they were prepared during our experiments with OpenCCM and ASSIST, thus they are well detailed but they also contain preliminary results and discussion of discontinued features.
   In the *release1* sub-directory, there are three final documents, part of the first release of this tool:
   1. ASSIST2CCM: Requirements
   2. ASSIST2CCM: User Manual
   3. ASSIST2CCM: Implementation

4. *Example*
   This directory stores some example ASSIST applications, to be used as a test for the tool:
   1. bench_dp (with synchronous and asynchronous events).
   2. prod_matrice, i.e. matrix multiply (with synchronous and asynchronous events).
   3. mandelbrot (with synchronous and asynchronous events, or RMI).
   4. regression tests: 1IN, 2IN_1OUT, 2IN_2OUT and bench_dp (1IN_1OUT) (with synchronous and asynchronous events).

5. *TASKCCM*
   For this directory, please refer to Paragraph 2.1.

2.1 The *TASKCCM directory*

Within the *taskccm* sub-tree, the user will find:

- The *seq* directory --> with tested programs written in C++ (sequential tests).
- The *solo_assist* directory --> it contains tested programs written in pure ASSIST.
- The *assist* directory --> it contains programs written in ASSIST, extended with the CCM modules *CCM_input* and *CCM_output.*
- The *script* directory --> it contains the configuration file for the environment variables (*env_CCM*), along with some scripts to compile the IDL3 files (*CCM_genera.sh, CCM_compila.sh, CCM_compila_main.sh*). It also stores scripts to launch and stop an application (*CCM_start.sh, CCM_stop.sh*), and a script to generate a CCM component starting from an ASSIST program (*assist2ccm.sh*).
- The *socket* directory --> it stores the code for *CDR_IO.h* and *CDRServer.java*
- The *code* directory --> it stores the file *demo.xml,* needed to compile the *main* code of the application, an example of a *main* file (*mymain.java*) and the relative XML file (*mymain.xml*).
- The *parserXML* directory--> here the user can find scripts that parse and modify the *ast.out.xml* file generated by ASSIST. They are used to reconfigure the ASSIST application.
- The *demo* directory --> it contains the regression test and further examples of CCM components.

2.1.1 Available Scripts

Within the *script* sub-directory, the user can find several packages, useful to deal with OpenCCM:

1. A script to configure the OpenCCM environment (*env_CCM*).

2. A script to compile the component IDL3 and Java source files (*CCM_genera.sh*, *CCM_compila.sh*, etc...).

3. Scripts to launch and stop a CCM application (*CCM_start.sh*, *CCM_stop.sh*).

4. Scripts to create the CCM component that wrap the ASSIST application (*assist2ccm.sh*)

2.1.1.1 The *env_CCM* Script

This script is used to configure the OpenCCM and OpenORB environment variables. This release features two configuration scripts, as used in our tests.

- *env_CCM_c1* --> to configure the environment on the *c1* cluster, where we installed OpenCCM version 0.8.0 and OpenORB version 1.3.1.

- *env_CCM_pianosa* --> to configure the environment on the *pianosa* cluster, where we installed OpenCCM version 0.8.1 and OpenORB version 1.4.0.BETA1.

The user needs to modify one of these scripts according to the configuration of his own cluster or machine. Then, the script can be added to the *.bashrc* file, so it is loaded every time the user logs in. The *bash source* command can be used to this purpose. A line like the following can be added to the *.bashrc* file.

```
source ${HOME}/TASKCCM/script/env_CCM
```

Within these two scripts:

- OpenCCM and OpenORB variables are set (*TCOO_HOME*, *CCM_DIR*, *OpenCCM_CONFIG_DIR*, *OpenCCM_HOMEDIR*, *OPENORB_LIB*, *TOOLS*, *PATH*),

- the Java *CLASSPATH* variable is set to include the directories where Java classes for the components can be found,

- some symbolic links are created to the customized versions of some scripts (*CCM_start.sh*, *CCM_stop.sh* and *demo.xml*).

2.1.1.2 Scripts to Compile Component IDL3 and Java Files

In the *taskccm* package, users can find some useful tools to simplify the compilation process:

- *CCM_genera.sh*
  This is used to compile the IDL3 files. In “ASSIST2CCM: Implementation”, more details on this script can be found. The name of the test to be compiled has to be provided as an input.

- *CCM_generaRMI.sh*
  This is similar to the previous one, except the fact that it does not make use of events.

- *CCM_compila.sh*
  CCM_*compila.sh* is used to compile stubs, skeletons and component source files. Its complete description is available in “ASSIST2CCM: Implementation”. The name of the test to be compiled has to be provided as an input.

- *CCM_compila_main.sh*
  CCM_*compila_main.sh* is used to compile the *main* body of the application. The name of the test to be compiled has to be provided as an input. It uses the templates available in the *code* directory: it copies the *demo.xml* file into the directory that contains *main*.

- *CCM_genera_eventi.sh*
  This script compiles both the IDL3 files with the event description and the Java files, generated starting from the IDL3. This is automatic, because no changes are needed in the Java files.

2.1.1.3 Scripts to Launch and Stop the CCM Application

In the *taskccm* package, the scripts *CCM_start.sh_c1* and *CCM_start.sh_pianosa* are used to launch the CCM applications on our two clusters. Similarly, *CCM_stop.sh_c1* and *CCM_stop.sh_pianosa* are used to stop it. The difference between the two versions is simply in the hard-coded machine names. The user should modify them to adapt them to their configuration. These scripts have two parameters: the test name and the number of *ComponentServer*s to be launched (default is 2).

2.1.1.4 The *assist2ccm.sh S*cript

This script, starting from any ASSIST program, is able to generate the executable code for the CCM wrapper component and for the ASSIST program extended with the *CCM_input* and *CCM_output* modules. In “ASSIST2CCM: Implementation”, more details on this script can be found.

It can be launched with 2, 3, or 4 parameters:

1. the first parameter is the name of the ASSIST file to be analyzed.

2. The second one is the name of the ASSIST module (sequential, parmod or generic) that the user wants to be wrapped with CCM. This is also the name of the CCM module generated by the tool.

3. The third parameter (optional) let the user choose between a synchronous (1) or asynchronous (0) component implementation. Default is asynchronous. It is not possible to use a synchronous implementation when the number of input events/streams is different from the number of output events/streams.

4. The fourth parameter (optional) is the name of the ASSSIST RC file (*ast_rc)* to be passed to the ASSIST compiler. If this file is not specified, then the default *ast_rc* will be used, that is the one available in the *$HOME* directory.

2.1.2 Tested Programs (*assist*/*solo_assist*/*seq*)

Three directories of tested examples are offered with *taskccm*:

- In the *seq* directory *seq,* users will find the source code of a sequential C++ implementation of *bench_dp*.

- In the *solo_assist* directory, users will find the source code of *bench_dp* and *prod_matrice*, written with pure ASSIST, and the used *ast_rc* file.

- In the *assist* directory, users will find the ASSIST code of *bench_dp* and *prod_matrice,* extended with the CCM modules *CCM_input* and *CCM_output*, along with the used *ast_rc* file.

2.1.3 The *CDR_IO* and *CDRServer* classes (*socket*)

In the *socket* directory, users will find the C++ *CDR_IO* class, used by ASSIST to exchange data with the Java threads that are part of the CCM component. We use sockets to this purpose. Also, they will find the Java *CDRserver* class, used by the Java threads of the CCM component to communicate with the *CCM_input* and *CCM_output* modules of the ASSIST program. More details are available in the document “ASSIST2CCM: Implementation”.

2.1.4 The XML Parser (*parserXML*)

Within the *parserXML* directory, we can find some scripts to modify the *ast.out.xml* file. They are used directly by the wrapping CCM component.

2.1.5 The *demo* Directory and Archived Tests (*demo, demo/archivio*)

The *demo* directory stores the source code for the CCM components.
Usually, a CCM application is made up of a component which wraps ASSIST (*bench_dpG*, *mandelbrotG*, *ev_moduloAssist*, *rmi_moduloAssist*...), a component managing events (*eventi*), some components which interact with the CCM-ASSIST component (*ev_moduloA*, *ev_moduloB*, ..., *client1*), and the component with the *main* (*mymain*).
For each component, our tool creates a new directory, to store IDL3 and source files.

In the *demo* directory, we also find the *archivio* (archive) sub-directory. Here, the user will find the regression tests (see Chapter 4) and many tested CCM applications. In order to run, these tests must be unpacked into the *demo* directory.

3 Using ASSIST2CCM

We summarize here how to run ASSIST2CCM version 1.0.

1. *Set the ASSIST2CCM environment variable*
   Set the *ASSIST2CCM* environment variable to point to the directory where the package is installed.

```
export ASSIST2CCM=$HOME/ASSIST2CCM
```

2. *Compile the analyzer tool*
   Perform *make* within the $ASSIST2CCM/analizzatore directory to compile the analyzer.

```
cd $ASSIST2CCM/analizzatore
make
```

3. *Customize the env_CCM script*
   Using the *env_CCM_c1* and *env_CCM_pianosa* scripts as an example (they are described in Paragraph 2.1.1.1), the user must write an *env_CCM* script to set up the work environment:

```
source env_CCM
```

4. *Customize CCM_start.sh and CCM_stop.sh*
   Again, using the example scripts (see Paragraph 2.1.1.3), the user will define two scripts, *CCM_start.sh* and *CCM_stop.sh*, customized with the names of the machines where the *NamingService* and the *ComponentServer*s will run.

5. *Run assist2ccm.sh*
   Now, we are ready to wrap an ASSIST program into a CCM component, by using the *assist2ccm.sh* script.

   1. Compiling using asynchronous events:

      assist2ccm.sh filename.ast componentG 0 file_ast_rc

   2. Compiling using synchronous events:

      assist2ccm.sh filename.ast componenteG 1 file_ast_rc

During its running, the *assist2ccm.sh* script will produce:

- A sub-directory named $*ASSIST2CCM*/*TASKCCM*/*demo*/*componenteG* (Figure 2), which contatins the IDL3 files of the component, its source code and the compiled output.
- A sub-directory named $*ASSIST2CCM*/*TASKCCM*/*demo*/*eventi*, if we are using events (Figura 2), which contains the IDL3 files for the events, the source code and the compiled output.
- The compiled ASSIST program, extended with the *CCM_input* and *CCM_output* modules, in the target directory chosen in the *ast*_rc file.

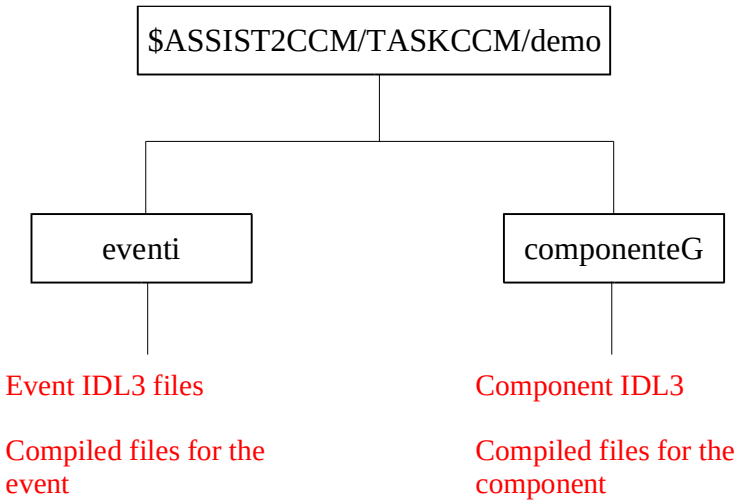

Figure 2. Directories created within the *demo* sub-tree by *assist2ccm.sh*.

4 Some Example CCM Applications: Regression Tests

In the $*ASSIST2CCM*/*TASKCCM*/*demo*/*archivio* sub-directory, the user will find the archived regression tests. Each test is a CCM application with a component that will be completely created by the *assist2ccm.sh* script, starting from an ASSIST program, and with one or more CCM components communicating (sending and receiving events) with the CCM component that wraps the ASSIST program.

4.1 REGRESSION_1IN

The first test is composed by two CCM components, the first (*ev_moduloA*) sends data to a second component (*solo_inG),* which wraps an ASSIST program. The code of this test is made up of:

- A directory named *assist_da_analizzare,* which contains the ASSIST program code *solo_in.ast.* Using the *assist2ccm.sh* script:

  assist2ccm.sh solo_in.ast solo_inG 0

  we generate and compile the used *events* (formerly, ASSIST streams) and the *solo_inG* component which wraps the ASSIST program.

- A directory named *ev_moduloA,* which contains the code (IDL3 and Java source code) of the *ev_moduloA* component.

- A directory named *mymain,* with the main code of the application.

4.2 REGRESSION_1IN_1OUT

The second test is composed by three CCM components, the first (*ev_moduloA*) sends data to a second component *(bench_dpG)*, which wraps an ASSIST program. The latter forwards them to a third component (*ev_moduloB*). The code of this test is made up of:

- A directory named *assist_da_analizzare,* which contains the ASSIST program code *bench_dp.ast*. Using the *assist2ccm.sh* script:

  asynchronous --> ***assist2ccm.sh bench_dp.ast bench_dpG 0***
  synchronous --> ***assist2ccm.sh bench_dp.ast bench_dpG 1***

  we generate and compile the used *events* (formerly, ASSIST streams) and the *bench_dpG* component which wraps the ASSIST program.

- Two directories, named *ev_moduloA* and *ev_moduloB,* which contain the code (IDL3 and Java source code) of the two components.

- A directory named *mymain,* with the main code of the application.

4.3 REGRESSION_2IN_1OUT

The third test is composed by four CCM components: two components (*ev_moduloA* and *ev_moduloC*) send data to a component *(bench_dpG)*, which wraps an ASSIST program. The latter forwards them to the last component (*ev_moduloB*). The code of this test is made up of:

- A directory named *assist_da_analizzare,* which contains the ASSIST program code *inout.ast*. Using the *assist2ccm.sh* script:

  asynchronous --> ***assist2ccm.sh inout.ast inoutG 0***
  synchronous --> ***assist2ccm.sh inout.ast inoutG 1***

  we generate and compile the used *events* (formerly, ASSIST streams) and the *inoutG* component which wraps the ASSIST program.

- Three directories, named *ev_moduloA, ev_moduloB* and *ev_moduloC,* which contain the code (IDL3 and Java source code) of the components.

- A directory named *mymain,* with the main code of the application.

4.4 REGRESSION_2IN_2OUT

The fourth test is composed by five CCM components: two components (*ev_moduloA* and *ev_moduloC*) send data to a component *(bench_dpG)*, which wraps an ASSIST program. The latter forwards them to two components (*ev_moduloB* and *ev_moduloD*). The code of this test is made up of:

- A directory named *assist_da_analizzare,* which contains the ASSIST program code *inout2.ast*. Using the *assist2ccm.sh* script:

  asynchronous --> ***assist2ccm.sh inout2.ast inout2G 0***
  synchronous --> ***assist2ccm.sh inout2.ast inout2G 1***

  we generate and compile the used *events* (formerly, ASSIST streams) and the *inout2G* component which wraps the ASSIST program.

- Four directories, named *ev_moduloA, ev_moduloB, ev_moduloC,* and *ev_moduloD,* which contain the code (IDL3 and Java source code) of the components.

- A directory named *mymain,* with the main code of the application.

5 Performance

Here, we present some experiments performed to measure the performance of ASSIST, when wrapped within a CCM component. Our tests were performed on a cluster (*pianosa*) composed of 23 Intel Pentium III Mobile, 800 Mhz, with 1 Gbyte RAM and dedicated Fast Ethernet network cards.
We tested both the RMI implementation and the event implementation. In the first case, we used OpenCCM 0.8.1 and OpenORB 1.4.0.BETA1; in the secon case, we used OpenCCM 0.8.1 and OpenORB 1.3.1.

The ASSIST benchmark we used is a synthetic data-parallel algorithm, which works on very large matrices (integer matrices, 350x350 elements, N=350). The task consists of running a *parallel for* loop N/10 times. The body of the loop is a transcendental function.

5.1 Performance: Events

Figure 3. Structure of the test CCM application.

The test is made up of three CCM components:

1. The first, generates an input matrix, and sends 10 requests using the event mechanism.
2. The second, which wraps the ASSIST program, accepts the requests and send results to the third.
3. The third collects the results.

In the following, we will call the cluster nodes *u3...u24*. The CCM and ASSIST processes are allocated as follows:

- *CCM_input* (Assist), *CCM_output* (Assist) and the second component (CCM) are on node *u4*
- the *OSM* process and one *VPM* (Assist) are on node *u6*
- the first and the third component (CCM) are on node *u23*
- the other VPMs (ASSIST) are allocated one on each of the unused nodes

In the following tables, we describe the timing of the sequential running and of the CCM applications, varying the number of virtual processors for the ASSIST module.

SEQUENZIALE
717.96

Table 4. Sequential timing of the benchmark.

NVPM	Bench_dp
1	733.46
2	369.66
4	197.12
8	102.06
16	62.50

Figure 5. Timing of the CCM application.

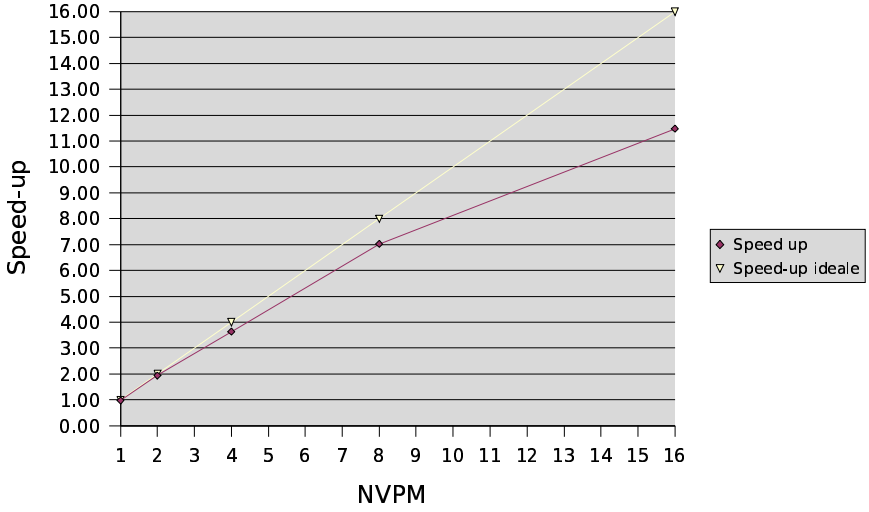

Figure 6. Speed-up of the CCM application.

Below, we report the bandwidth that we can reach by changing the granularity of communications.

Message size	Bandwidth byte/sec
0.3	96889
1.0	410899
4.0	934966
16.0	2811873
64.0	4325950
256.0	3584097
511.9	3544677
514.7	3530324
1024.0	4374958

Figure 7. Bandwidth with the event implementation.

5.2 Performance: RMI

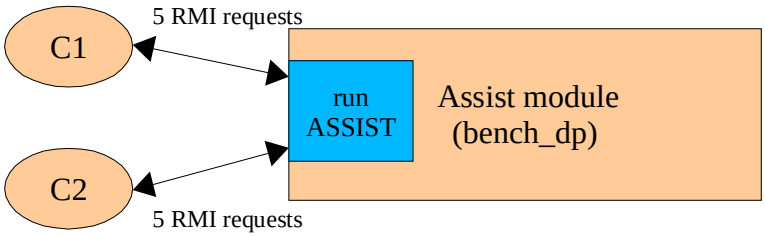

Figure 8. Structure of the CCM application.

The test is made up of three components:

1. The first and the second component (C1 and C2) send their requests to the Assist module, using the RMI method *esegui_assist*. The total number of requests is 10. Requests from each client are executed sequentially, so, in order to exploit communication/computation parallelism within the wrapped application, more than one client is needed. Two clients are enough in this settings to reach good speed-up.
2. The third component wraps the ASSIST program. It accepts requests from the first and the second component, elaborates them and sends the results back.

In the following, we will call the cluster nodes *u3...u24*. The CCM and ASSIST processes are allocated as follows:

- *CCM_input* (Assist), *CCM_output* (Assist) and the second component (CCM) are on node *u12*
- the *ISM* process and one *VPM* (Assist) are on node *u13*
- the *OSM* process and one *VPM* (Assist) are on node *u14*
- the client components *ClientA* and *ClientB* (CCM) are on node *u23*
- the other VPMs (ASSIST) are allocated one on each of the unused nodes

In the following tables, we describe the timing of the sequential running and of the CCM applications, varying the number of virtual processors for the ASSIST module.

SEQUENZIALE
717.96

Table 9. Sequential timing of the benchmark.

NVPM	Bench_dp
1	649.08
2	326.40
4	172.95
8	88.19
16	55.69

Table 10. Timing of the test CCM application.

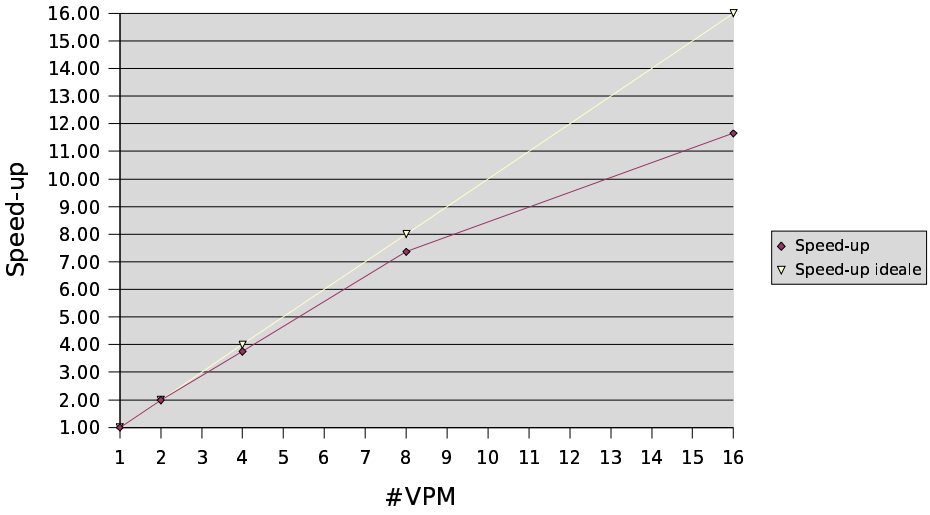

Table 11. Speed-up of the CCM application.